Questions

1. Assuming that the following JOS kernel code is correct, what type should variable x have, uintptr_t or physaddr_t?
    a. uintptr_t
2. There is only one page directory filled so far. What are the corresponding pml4 and pdpe entry (row) numbers for this page directory? What entries (rows) in the page directory have been filled in at this point? What addresses do they map and where do they point? In other words, fill out this table as much as possible:

Entry	Base Virtual Address	Points to (logically):
511	?	Page table for top 2MB of phys memory
510	?	?
.	?	?
.	?	?
.	?	?
159	?	?
5	?	?
1	0x8000200000	?
0	0x8000000000	?

3. We have placed the kernel and user environment in the same address space. Why will user programs not be able to read or write the kernel's memory? What specific mechanisms protect the kernel memory?
    a. The permission bits protect the kernel memory at each page – the user doesn't have access without the right permissions – PTE_W and PTE_U.
4. What is the maximum amount of physical memory that this operating system can support? Why?
    a. 256MB according to kern/pmap.h – only the 256mb is mapped, so that is all that's supported.
    b. If we aren't considering what's mapped and instead the theoretical max: then 512 entries at each level of directory, each entry representing 4096 bytes: 512^4 * 4096 bytes in total memory: ~280TB max.
5. How much space overhead is there for managing memory, if we actually had the maximum amount of physical memory? How is this overhead broken down?
    a. 8MB
6. Read the simple page table setup code in kern/bootstrap.S.
   The bootloader tests whether the CPU supports long (64-bit) mode. It initializes a simple set of page tables for the first 4GB of memory. These pages map virtual addresses in the lowest 3GB to the same physical addresses, and then map the upper 256 MB back to the lowest 256 MB of memory. At this point, the bootloader places the CPU in long mode. Note that our bootloader transitioning to long mode isn't strictly necessary; typically, a bootloader only runs in long mode to load a 64-bit kernel at a high (>4 GB) virtual memory address.

Note that once we transfer control to the kernel, the kernel assumes the CPU supports 64-bit mode. Assuming the kernel was loaded in the lower 4GB of virtual address space, the kernel itself could test whether the CPU supports long mode and determine dynamically whether to run in 64 or 32-bit mode. Of course, this would substantially complicate the boot process.